NASA Technical Memorandum 4451

Parallel Solution of High-Order Numerical Schemes for Solving Incompressible Flows

Edward J. Milner, Avi Lin,
May-Fun Liou, and Richard A. Blech

APRIL 1993

NASA

NASA Technical Memorandum 4451

Parallel Solution of High-Order Numerical Schemes for Solving Incompressible Flows

Edward J. Milner
Lewis Research Center
Cleveland, Ohio

Avi Lin
University of Pennsylvania
Philadelphia, Pennsylvania

May-Fun Liou and Richard A. Blech
Lewis Research Center
Cleveland, Ohio

NASA
National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Program
1993

Parallel Solution of High-Order Numerical Schemes for Solving Incompressible Flows

Edward J. Milner
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

Avi Lin
University of Pennsylvania
Philadelphia, Pennsylvania 19104

May-Fun Liou and Richard A. Blech
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135

Summary

A new parallel numerical scheme for solving incompressible steady-state flows is presented. The algorithm uses a finite-difference approach to solving the Navier-Stokes equations. The algorithms are scalable and expandable. They may be used with only two processors or with as many processors as are available. The code is general and expandable. Any size grid may be used. Four processors of the NASA Lewis Research Center Hypercluster were used to solve for steady-state flow in a driven square cavity. The Hypercluster was configured in a distributed-memory, hypercube-like architecture. By using a 50-by-50 finite-difference solution grid, an efficiency of 74 percent (a speedup of 2.96) was obtained.

Introduction

Over the years, as thrust-to-weight ratio has increased, propulsion systems have evolved into very sophisticated designs, requiring intricate interaction among an enormous number of complicated components. This complexity is the result of trying to harness as much power as possible from a propulsion system of given size and weight.

As thrust-to-weight ratio continues to increase in the future, the propulsion system designer will have to rely heavily on high-speed computer simulations. Today, the simulation of a modern propulsion system on state-of-the-art serial computers may consume hundreds of hours of execution time to achieve the level of detail required. Yet, to be of help to the propulsion system designer, simulation results must be available in a reasonable amount of time.

Because for many years simulation complexity has been increasing faster than the mainframe computer speeds capable of handling it, other techniques have been investigated to reduce the effective wallclock time to completion. Parallel processing has gained considerable popularity in the last few years, and much new commercial hardware is available today. But the concept of having several computers working in unison on the same problem has been around for a long time. For example, during the early 1980's researchers at the NASA Lewis Research Center demonstrated the feasibility of parallel processing in engine simulation (refs. 1 through 5). The researchers executed a small helicopter engine simulation by using four processors in unison on a parallel processing system developed in-house at NASA Lewis (ref. 6).

A characteristic of parallel processing that has become very apparent today is that hardware is leading the software available by a considerable amount. Powerful hardware is commercially available, but the software to harness that powerful hardware is not. This is particularly apparent in the area of fluid dynamics. Fluid dynamics problems are generally very computationally intensive, requiring solutions of systems of partial differential equations over huge one-, two-, and three-dimensional grids. One characteristic of these problems is that they often require solution of block-diagonal systems over finite-difference grids.

This report discusses techniques for rapidly solving the steady-state, incompressible Navier-Stokes equations by using parallel processing and finite-differencing techniques. Algorithms are presented that are scalable, general, and portable. The algorithms can be used with as few as two processors or, for a large enough problem, with as many processors as are available. Any size problem, within the limits of the computer's memory, can be solved. Care was

taken to minimize the effects of machine-specific items in order to ease porting of the code between multiple-input, multiple-data (MIMD) distributed-memory computers. Machine-specific items, such as COMMON and vector processing, are not used in the code. For demonstration purposes the results that were obtained from applying the algorithms to three classical fluid dynamics problems are presented, including solving for the steady-state flow within a driven square cavity. Finally, a technique for making the algorithms even faster is discussed.

Algorithm Overview

With the aggressive advances taking place in parallel processing hardware, new software and algorithms must be developed to harness this increase in computing capability. Because of the complicated nature of fluid dynamics problems many studies are attempting to develop new techniques and algorithms for taking advantage of parallelism in these computationally intensive problems. One such technique is the finite-difference algorithm that was developed by A. Lin for solving the steady-state incompressible Navier-Stokes equations. This algorithm, which has the potential to provide a scalable, parallel solution, has been well documented over the past several years (refs. 7 through 10). For convenience, the key features of the algorithm are highlighted here. For a formal and detailed development of these concepts, consult the references.

The steady-state incompressible Navier-Stokes equations can be written in the dimensionless form:

Continuity equation:

$$u_x + v_y = 0 \tag{1}$$

x-momentum equation:

$$uu_x + vu_y + p_x = \frac{1}{\mathrm{Re}} \nabla^2 u \tag{2}$$

y-momentum equation:

$$uv_x + vv_y + p_y = \frac{1}{\mathrm{Re}} \nabla^2 v \tag{3}$$

where $g_x = \partial g/\partial x$, p is the pressure function, u is the x-component of velocity, v is the y-component of velocity, and Re is the Reynolds number. This is a nonlinear system of elliptic partial differential equations that, in general, must be solved iteratively.

Define χ, the vector of variables, as

$$\chi = \begin{pmatrix} u \\ v \\ p \end{pmatrix} \tag{4}$$

Let us also choose the following convention. Let χ be defined as the values at the ith iteration level. Let χ^* be the values at the $(i-1)$th iteration level. Hence u is the x-component of velocity at the ith iteration level and u^* is the x-component of velocity at the $(i-1)$th iteration level.

If we define the operators

$$\mathbf{C} = u^* \frac{\partial}{\partial x} + v^* \frac{\partial}{\partial y} \tag{5}$$

and

$$\mathbf{L} = \mathbf{C} - \frac{1}{\mathrm{Re}} \nabla^2 \tag{6}$$

and apply the following second-order quasi-linearization, given the general term $f(x)g(x)$,

$$fg \approx f^* g + g^* f - f^* g^* \tag{7}$$

Then the system of equations shown in equations (1) through (3) can be solved iteratively as

$$\Lambda\chi = \eta\chi^* \tag{8}$$

where

$$\Lambda = \begin{pmatrix} K\frac{\partial}{\partial x} & K\frac{\partial}{\partial y} & 0 \\ \mathbf{L} + u_x^* & u_y^* & \frac{\partial}{\partial x} \\ v_x^* & \mathbf{L} + v_y^* & \frac{\partial}{\partial y} \end{pmatrix} \tag{9}$$

and

$$\eta = \begin{pmatrix} 0 & 0 & 0 \\ \mathbf{C} & 0 & 0 \\ 0 & \mathbf{C} & 0 \end{pmatrix} \tag{10}$$

where K is a constant with the same units as the velocity. The continuity equation is multiplied by K in order to bring it to the same units as the momentum equations to ease eigenvalue analysis.

The iteration matrix for equation (8) is

$$\xi = \Lambda^{-1}\eta \tag{11}$$

All the eigenvalues of ξ for the continuous problem are bounded by 1 for all values of $K \neq 0$ when χ^* is sufficiently close to χ.

Equation (8) can be solved by using the finite-difference numerical scheme shown in figure 1. Note that the primitive variables are staggered over the finite-difference grid. Boundary conditions for the internal flow field are also shown in the figure. Usually some artificial computational cells are added by appending two adjacent boundaries (i.e., $i = 1$ and $j = 1$). This is done to make the application of the algorithm easier. As shown in the figure, along the left column of cells appended,

$$(v_{1,j} + v_{2,j})/2 = v_{b_j} \tag{12}$$

where b_j denotes the jth boundary value of the y-component of velocity. Along the bottom row of cells appended,

$$(u_{i,1} + u_{i,2})/2 = u_{b_i} \tag{13}$$

where b_i denotes the ith boundary value of the x-component of velocity. The pressure is zero within all the cells appended. The pressure variable does not appear on the boundary because of the nature of the Navier-Stokes equations, as discussed in references 9 and 10.

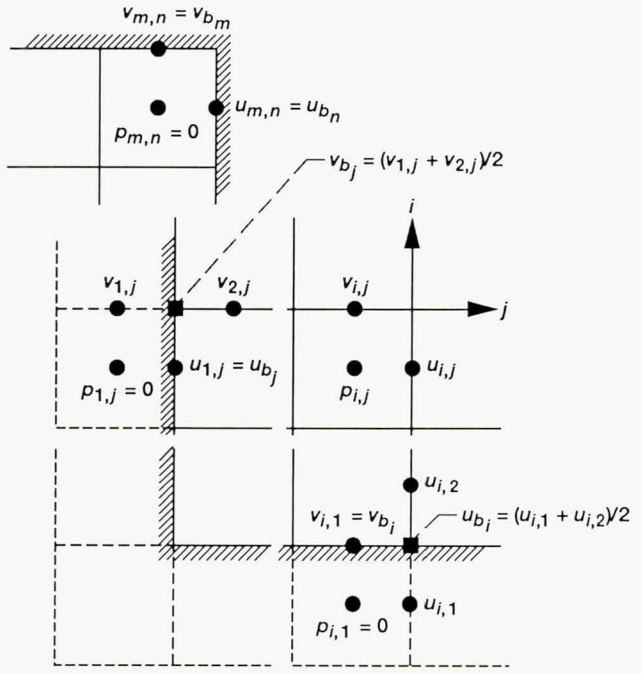

Figure 1.—Variable locations and internal flow boundary conditions, where p is pressure function, u is x-component of velocity, v is y-component of velocity, u_b is boundary value of x-component of velocity, and v_b is boundary value of y-component of velocity.

Let us define the grid and flow parameters

$$\alpha = \frac{h}{k} \qquad \text{and} \qquad \beta = \frac{1}{\text{Re}}\,\frac{1}{h} \tag{14}$$

where h is the finite-difference grid spacing in the i- or x-direction and k is the finite-difference grid spacing in the j- or y-direction.

A possible finite-difference approximation for the system given by equation (8) can be expressed as follows: The discrete continuity equation is

$$\left(u_{i,j} - u_{i-1,j}\right) + \alpha\left(v_{i,j} - v_{i,j-1}\right) = 0 \tag{15}$$

And the discrete momentum equations are

$$\begin{aligned} h\bar{\mathbf{C}}\left(u_{i,j}\right) &+ \left(p_{i+1,j} - p_{i,j}\right) - \beta\left(u_{i+1,j} - u_{i-1,j}\right) \\ &- \beta\alpha^2\left(u_{i,j+1} - u_{i,j-1}\right) + 2\beta(1 + \alpha^2)u_{i,j} \\ &+ h\left[\left(u_x^*\right)_{i,j} u_{i,j} + \left(u_j^*\right)_{i,j} v_{i,j}\right] = 0 \end{aligned} \tag{16}$$

and

$$\begin{aligned} h\bar{\mathbf{C}}\left(v_{i,j}\right) &+ \alpha\left(p_{i,j+1} - p_{i,j}\right) - \beta\left(v_{i+1,j} - v_{i-1,j}\right) \\ &- \beta\alpha^2\left(v_{i,j+1} - v_{i,j-1}\right) + 2\beta\left(1 + \alpha^2\right)v_{i,j} \\ &+ h\left[\left(v_j^*\right)_{i,j} v_{i,j} + \left(v_x^*\right)_{i,j} u_{i,j}\right] = 0 \end{aligned} \tag{17}$$

where $\bar{\mathbf{C}}$ is the second-order, upwind, finite-difference scheme of the convection operator $\mathbf{C}$ given in equation (5). In evaluating $\bar{\mathbf{C}}$, the finite difference of the first derivative is taken as a forward difference if the variable is negative ($(\phi_x)_i = (\phi_{i+1} - \phi_i)/h$ or $(\phi_y)_j = (\phi_{j+1} - \phi_j)/k$) and as a backward difference if the variable is positive ($(\phi_x)_i = (\phi_i - \phi_{i-1})/h$ or $(\phi_y)_j = (\phi_j - \phi_{j-1})/k$). This is done to ensure the diagonal dominance of the system.

Equations (1) through (3) depend on the change in pressure, not the pressure level itself. Thus the pressure level must be set. This can be done by enforcing a fixed pressure level in one of the cells. Although any cell may be chosen, a boundary or corner cell is generally chosen for convenience (see fig. 1). Equations (15) through (17) represent a second-order approximation to the Navier-Stokes equations. They reflect the fact that the total mass flux through the domain Ω is preserved. The scheme allows for the conservation of mass at each

computational cell at every iteration level. The conservation of mass is an important condition of steady-state incompressible flow and should be maintained.

Parallel Environment and Computation Scheme

As shall be seen later in this section when the parallel computation scheme is discussed, the algorithm highlighted previously lends itself quite nicely to solution in a parallel environment. The parallel environment chosen for this study was the Hypercluster, a versatile MIMD parallel processing testbed that has been developed at NASA Lewis for algorithm and architecture research (refs. 11 and 12). This section focuses on describing the Hypercluster architecture and the parallel computational scheme for solving equation (8).

Hypercluster Architecture

The Hypercluster parallel environment is an experimental machine that has been specifically designed to perform well on computational fluid dynamics (CFD) problems. The rationale behind the design of the Hypercluster testbed evolved from the complexity of investigating CFD algorithms on parallel and vector hardware. This rationale is discussed in reference 11. A four-node Hypercluster consists of four clusters of processors that are interconnected in a binary n-cube configuration by internode links. The cluster of processors at a node may consist of any number of scalar and/or vector processors. The architecture for one such four-node prototype is shown in figure 2. The links allow communication between nodes and consist of two control processors (CP's) communicating through dual-ported memory. It is important to note that the CP's can route information throughout the Hypercluster without interrupting computation of the current application. Thus, communication occurs in parallel with computation.

The Hypercluster resembles other hypercube architectures in many respects but has the following important differences:

(1) Each node of the Hypercluster can consist of more than one scalar and/or vector processor.

(2) Processors within a node communicate through shared memory.

(3) Independent CP's perform the internode communication without interrupting the processors within a node that are performing the application computations. These CP's are programmable, allowing investigation of various message-passing protocols.

(4) The processor technology within each node is not limited to any particular vendor. The use of a standard bus allows any processor board that is available for the bus to be incorporated. This feature also provides a rapid method for upgrading the system hardware.

The user interacts with the hardware through a menu-driven Hypercluster operating system (HYCLOPS), which runs on the front-end processor. Hypercluster programs are written in Fortran 77, with library support for parallel and vector processing. Each processor in the Hypercluster runs a message-passing kernel that supports interprocessor communication. The kernel supports distributed-memory communication through simple "send" and "receive" library calls. Shared-memory communication is supported through COMMON blocks. Details on the Hypercluster programming and operating environment can be found in reference 12.

The Hypercluster provides a versatile environment that allows experimentation with many different subset architectures and programming styles. Because of the interest in distributed-memory systems this mode of Hypercluster operation was selected for the test cases presented herein.

Parallel Computation Scheme

An ideal application of the numerical scheme just described is to solve linearized systems iteratively. For any grid point (i,j) the following linear relationship holds:

$$\mathbf{P}_{i,j}\chi_{i,j} + \mathbf{W}_{i,j}\chi_{i-1,j} + \mathbf{N}_{i,j}\chi_{i,j+1} + \mathbf{E}_{i,j}\chi_{i+1,j} + \mathbf{S}_{i,j}\chi_{i,j-1} = (\mathrm{RHS})_{i,j} \tag{18}$$

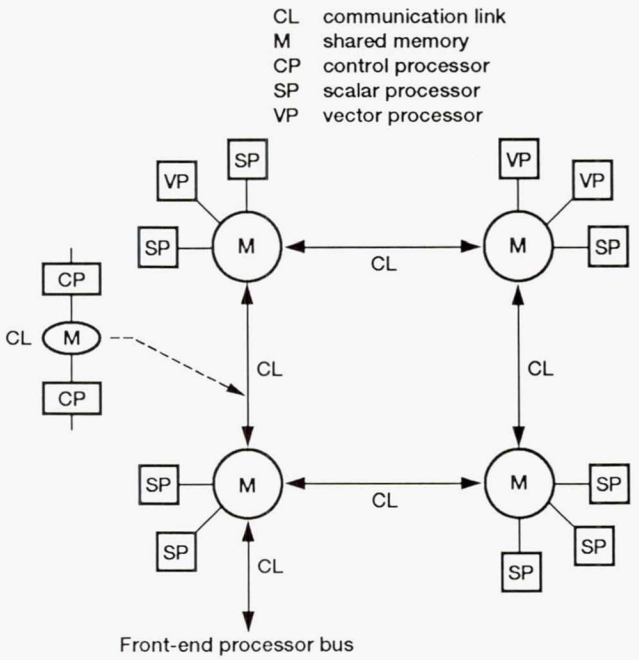

Figure 2.—Two-dimensional Hypercluster configuration.

where the coefficient matrices **P**, **W**, **N**, **E**, and **S** are derived from the operator Λ by using the discrete finite-difference equations (i.e., equation (9) together with equations (15) through (17)), χ is the value of $(u, v, p)^T$ at the kth iteration level, and RHS is a known term that depends on $\chi^{(k-1)}$, or the values at the $(k-1)$th iteration level.

Several parallel iterative methods for solving equation (18) have been discussed previously (refs. 13 and 14). The iterative approach that follows, which is called line relaxation, is well suited for use with distributed-memory MIMD parallel computers because communication complexity is minimal.

The iteration equation that is considered by the line-relaxation iterative algorithm is the following block-tridiagonal system:

$$A_r\chi_{r-1} + B_r\chi_r + C\chi_{r+1} = D_r,\ 1 \le r \le n \qquad (19)$$

This system is obtained from equation (18) by deciding upon the implicit line, either i or j, moving the off-line terms to the right-hand side of the equation, and combining them with the RHS term to form the D term. The parallel algorithm for solving equation (19) is based on a more general approach that is discussed in reference 15.

The scheme for solving equation (19) is as follows: If q processors are available to solve the system, split the n equations into q strips. Accomplish this by identifying q internal equations, or key equations. Denote by key_i the location of these equations so that $1 < \text{key}_1 < \text{key}_2 < \ldots < \text{key}_q < n$. For the remainder of this discussion, the following convention is used: Denote by the rth strip the set of equations from the $(\text{key}_{r-1}+1)$ equation through the $(\text{key}_{r+1}-1)$ equation. For completeness, let $\text{key}_0 = 0$ and $\text{key}_{q+1} = (n+1)$.

Note that there is now some overlapping of the equations in the strips. For example, as shown in figure 3, the rth and the $(r-1)$th strips have the equations $(\text{key}_{r-1}+1)$ through (key_r-1) in common. Likewise, the rth and the $(r+1)$th strips have the equations (key_r+1) through $(\text{key}_{r+1}-1)$ in common. As explained in the discussion that follows, the reason for this overlapping is to get a block-tridiagonal relationship between $\chi_{\text{key}_{i-1}}$, χ_{key_i}, and $\chi_{\text{key}_{i+1}}$ for each processor i used in the parallel solution of the problem.

The parallel solution of equation (19) takes place in the following fashion: Since we have q processors to solve equation (19), we assign to the rth processor, $1 \le r \le q$, the rth strip of equations, namely, the equations $(\text{key}_{r-1}+1)$ through $(\text{key}_{r+1}-1)$.

For the moment consider the equations on processor 1. The system of equations is as shown in figure 4. Multiplying the first equation by $A_2B_1^{-1}$ and subtracting from the second equation, we see that the A_2 coefficient in the second equation is eliminated and the C_2 coefficient is left alone. In addition, the B_2 and D_2 coefficients are replaced with the modified coefficients B_2' and D_2'. If we repeat the process and multiply this new second equation by $A_3(B_2')^{-1}$ and subtract from the third equation, we eliminate the A_3 coefficient and get new values for the B_3 and D_3 coefficents. Continue repeating this process (key_1-1) times until coefficient A_{key_1} is eliminated.

Now go to the last equation on the processor. Multiply this equation by $C_{\text{key}_2-2}\left(B_{\text{key}_2-1}\right)^{-1}$ and subtract the result from the second-last equation. The C_{key_2-2} coefficient is eliminated and the A_{key_2-2} coefficient is left alone. Coefficient B_{key_2-2} is replaced with a new-valued coefficient, and a new coefficient $C'_{\text{key}_2-2} = -C_{\text{key}_2-2}\left(B_{\text{key}_2-1}\right)^{-1}C_{\text{key}_2-1}$ is introduced. This coefficient multiplies χ_{key_2} and this term will carry along in each step as we repeat the process to eliminate the C coefficients back to the key equation. Doing so, the key equation on processor 1 will be transformed into an equation of the form

$$B^*_{\text{key}_1}\chi_{\text{key}_1} + C^*_{\text{key}_1}\chi_{\text{key}_2} = D^*_{\text{key}_1} \qquad (20)$$

Let us now turn our attention to processors 2 through $(q-1)$. The system of equations on processor 2 is as shown in figure 5. The same process is used to reduce this set of equations as was used on processor 1. The only difference is that in the

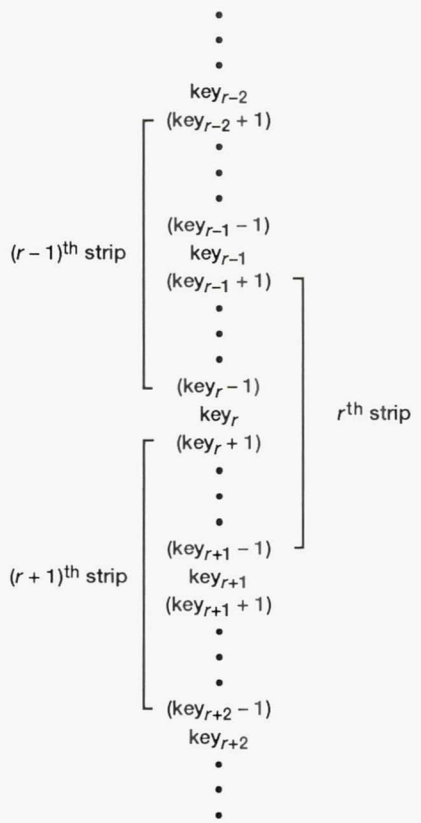

Figure 3.—Calculation strip layout.

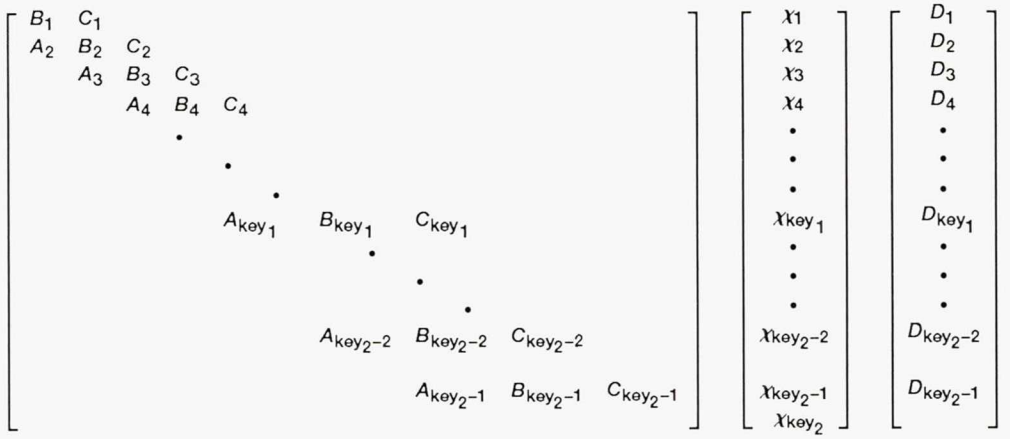

Figure 4.—Processor 1 equations.

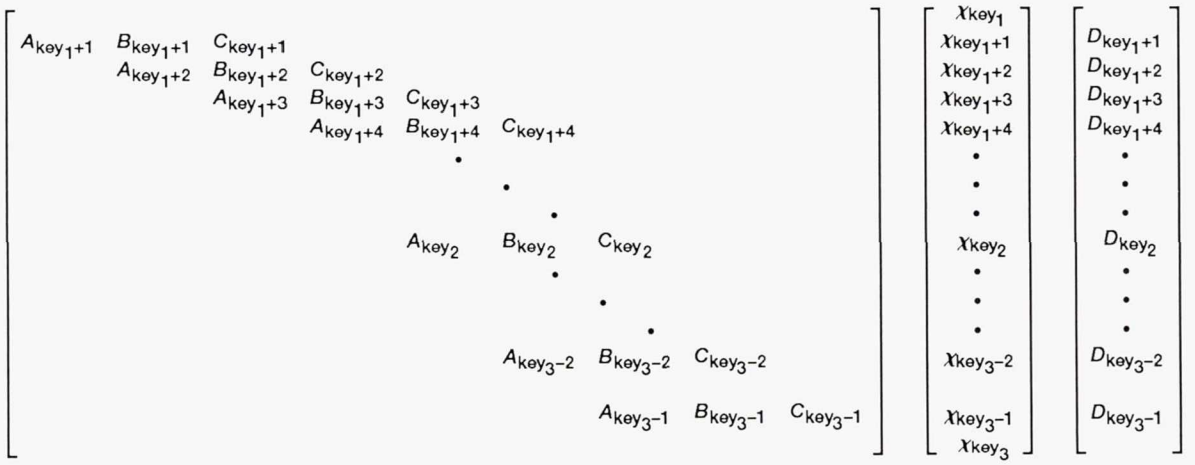

Figure 5.—Processor 2 equations.

first equation the A_{key_1+1} coefficient is not zero. Therefore, as we reduce the equations forward toward equation key_2, a term gets carried along that multiplies χ_{key_1}. Hence, when the operations are completed on processor 2, the key equation on that processor will be of the form

$$A^*_{\text{key}_2}\chi_{\text{key}_1} + B^*_{\text{key}_2}\chi_{\text{key}_2} + C^*_{\text{key}_2}\chi_{\text{key}_3} = D^*_{\text{key}_2} \qquad (21)$$

In general, for the ith processor, $2 \le i \le (q-1)$, the reduced key equation will be of the form

$$A^*_{\text{key}_i}\chi_{\text{key}_{i-1}} + B^*_{\text{key}_i}\chi_{\text{key}_i} + C^*_{\text{key}_i}\chi_{\text{key}_{i+1}} = D^*_{\text{key}_i} \qquad (22)$$

The system of equations on the last processor is as shown in figure 6. Notice that when we begin to perform the backward reduction (i.e., multiply the last equation by $C_{n-1}B_{n-1}$ and subtract the result from the second-last equation), the C_{n-1} coefficient is eliminated. Hence, the reduced key equation on this processor is of the form

$$A^*_{\text{key}_q}\chi_{\text{key}_{q-1}} + B^*_{\text{key}_q}\chi_{\text{key}_q} = D^*_{\text{key}_q} \qquad (23)$$

Equations (20) through (23) form a block-tridiagonal system, whose solution is the solution to equation (19) at the key points, key_i, $1 \le i \le q$. This is a system of order q and, because it is usually a very small system (q is usually a small number), it can be solved on a single processor by using standard serial algorithms. However, if q were large, the system could be solved in parallel by applying the same techniques that we applied to equation (19). After the solution is obtained at the key points, these values are sent back to the appropriate processors.

Now if the solution of equation (19) were required at only a few points, and those points were included as key points, we would be done. We would have the solution for equation (19) at the points desired, and we would not be required to perform a back substitution to obtain the remaining portion of the solution of equation (19). However, to obtain the entire solution of equation (19), each processor, i, uses appropriate key point results to solve for $\chi_{\text{key}_{i-1}+1}$ through χ_{key_i-1} by way of back

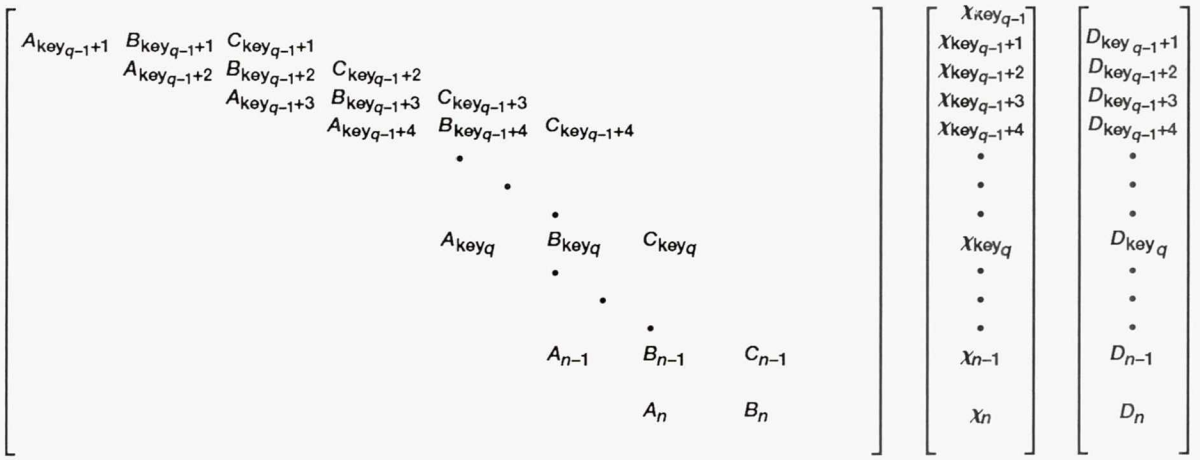

Figure 6.—Last processor equations.

substitution. An exception is the last processor, processor q, which solves for $\chi_{key_{q-1}+1}$ through χ_n.

The algorithm described here is both scalable and expandable. The work load can be designed to be well balanced by choosing key points appropriately. For example, when a parallel homogeneous system is used, all the calculation strips generally will be chosen to be the same size. This will approach a balanced work load among the processors. The work load for the back-substitution phase is balanced among all the processors, except for the last one. During the back substitution this processor has to execute approximately double the number of substitutions as any of the other processors.

Results and Discussion

Speedup results that were obtained from applying the parallel algorithm to several different fluid dynamics problems are presented in this section. Two special flow fields with known results (ref. 16) were selected for evaluating the parallel algorithm, including the quasi-linear approach that was used in the algorithm development. These first two applications demonstrate the fast convergence rate of the parallel algorithm. They also show that the speedup which was realized increases as relative communication time decreases. Following these demonstrations, the parallel algorithm is used to solve a driven square cavity. This well-known problem displays the intricate, coupled-flow relationships that are found in complicated fluid dynamics problems.

The first problem we consider is examining the boundary layer along a flat plate at zero pressure gradient. The governing equations are

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \frac{1}{\mathrm{Re}}\frac{\partial^2 u}{\partial y^2} \tag{24}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{25}$$

with the boundary conditions

$$y = 0: \quad u = v = 0; \qquad y = \infty: \quad u = U_\infty \tag{26}$$

where u and v are the x-component and y-component of velocity, respectively; and U_∞ is the free-stream velocity as shown in figure 7. As discussed in reference 16 (pp. 125 and 126), by letting f be the normalized stream function, the governing equations become Blasius' equation, which is

$$ff'' + 2f''' = 0 \tag{27}$$

with the boundary conditions

$$f(0) = f'(0) = 0, \qquad f'(\infty) = 1 \tag{28}$$

The location of the infinity point, a sufficiently large number, was arbitrarily chosen to be 8, as also was the solution grid of 1000 points. In only six iterations the solution is converged to a residual of the order 10^{-12} (see table I). As shown in

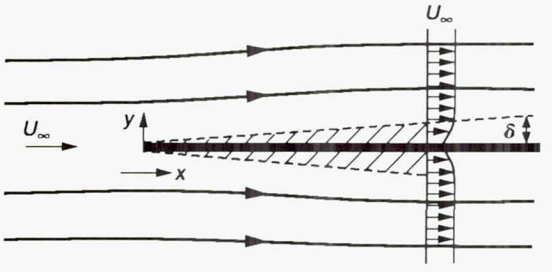

Figure 7.—Boundary layer along flat plate at zero incidence.

TABLE I.—RESIDUAL ERROR VERSUS NUMBER OF ITERATIONS FOR BOUNDARY LAYER PROBLEM

Iteration	Residual (maximum) error
1	0.14258×10^{1}
2	.26890
3	$.28632\times10^{-1}$
4	$.24873\times10^{-3}$
5	$.16154\times10^{-7}$
6	$.11084\times10^{-11}$

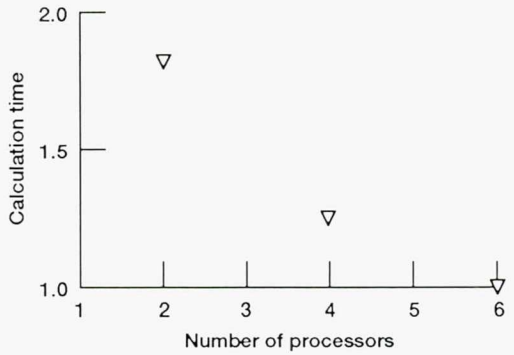

Figure 8.—Calculation time versus number of processors for boundary layer problem.

figure 8, going from two processors to six processors did not give the theoretical increase of 3 in execution speed. However, this result was expected. Because of the simple nature of the problem the calculation time per processor was quite short, and hence the communication time between processors was not insignificant relative to the short calculation time per processor.

The second problem examined was flow near a disk rotating in a fluid at rest. A diagram of the problem is shown in figure 9. A layer of fluid is carried by the disk owing to the action of viscous forces. The centrifugal forces in the layer give rise to secondary flow that is directed radially outward. The governing equations, as given in reference 16 (p. 93), are

$$u\frac{\partial u}{\partial r} - \frac{v^2}{r} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial r} + \frac{1}{\mathrm{Re}}\left\{\frac{\partial^2 u}{\partial r^2} + \frac{\partial}{\partial r}\left(\frac{u}{r}\right) + \frac{\partial^2 u}{\partial z^2}\right\} \tag{29}$$

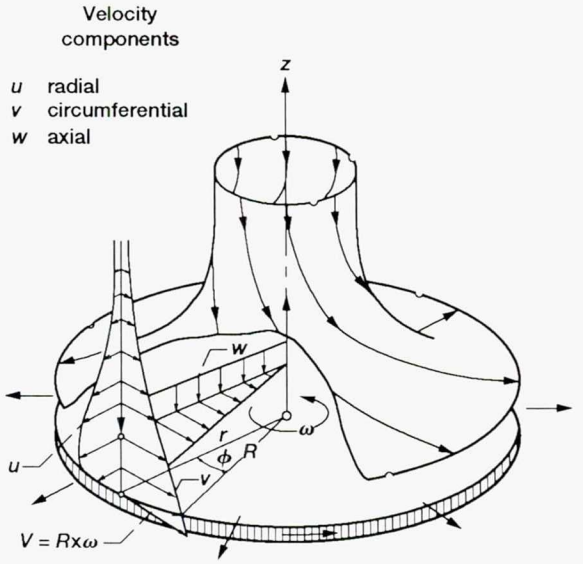

Figure 9.—Flow near disk rotating in fluid at rest.

$$u\frac{\partial v}{\partial r} + \frac{uv}{r} + w\frac{\partial v}{\partial z} = \frac{1}{\mathrm{Re}}\left\{\frac{\partial^2 v}{\partial r^2} + \frac{\partial}{\partial r}\left(\frac{v}{r}\right) + \frac{\partial^2 v}{\partial z^2}\right\} \tag{30}$$

$$u\frac{\partial w}{\partial r} + w\frac{\partial w}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \frac{1}{\mathrm{Re}}\left\{\frac{\partial^2 w}{\partial r^2} + \frac{1}{r}\frac{\partial w}{\partial r} + \frac{\partial^2 w}{\partial z^2}\right\} \tag{31}$$

$$\frac{\partial u}{\partial r} + \frac{u}{r} + \frac{\partial w}{\partial z} = 0 \tag{32}$$

where ρ is the density of the fluid, r is the radius of the disk, p is the pressure, and u, v, and w are the radial, circumferential, and axial velocity components, respectively. The Sparrow-Gregg equations governing this problem (ref. 16, p. 95) are

$$2F + H' = 0 \tag{33}$$

$$F^2 + F'H = 0 \tag{34}$$

$$G^2 - F'' = 0 \tag{35}$$

$$2FG + HG' - G'' = 0 \tag{36}$$

with boundary conditions

$$F(0) = H(0) = F(\infty) = G(\infty) = 0, \qquad G(0) = 1 \tag{37}$$

where F, G, and H are the normalized velocity components. The location of the infinity point was again arbitrarily chosen to be 8, and the solution grid was chosen to be 500 points.

The algorithm converges extremely fast. In only 10 iterations the residual is of the order 10^{-13} as shown in table II, but this is still a relatively simple problem for checking the parallel algorithm. However, now there is a little more substance to the calculations per processor relative to communication than there was for the previous test case, as reflected by figure 10. This time, as shown in the figure, a relative calculation speedup of more than 2 was realized in going from two processors to six processors.

The parallel iterative algorithm was then applied to a more complicated fluid dynamics problem, solving for steady-state flow within a driven square cavity. The cavity, which had three rigid wall surfaces as shown in figure 11, was filled with fluid. The fourth surface moved at constant velocity within its own plane, causing the fluid within the cavity to swirl. The defining equations are equations (1) through (3) with the boundary conditions that the velocity components u and v are zero everywhere except at the moving surface. At that surface the x-component of velocity, u, is 1 and the y-component of velocity, v, is 0. The swirling action can cause vortices to appear, as shown in the figure. The approach for solving this problem was similar to some other primitive variable approaches that have been devised for the two-dimensional driven cavity flow (ref. 17). As discussed earlier and shown in figure 1, the pressure, the velocity components, and the equations were staggered over the finite-difference grid. The convection terms were discretized in a special way to preserve the second-order stability scheme. Because of the complicated nature of the flows and the strong coupling within the cavity, convergence was slow.

TABLE II.—RESIDUAL ERROR VERSUS NUMBER OF ITERATIONS FOR DISK ROTATING IN FLUID AT REST

Iteration	Residual (maximum) error
1	0.15019×10^{2}
2	$.52285\times10^{1}$
3	$.23837\times10^{1}$
4	$.10327\times10^{1}$
5	.39308
6	$.94527\times10^{-1}$
7	$.47686\times10^{-2}$
8	$.14872\times10^{-4}$
9	$.78692\times10^{-9}$
10	$.60396\times10^{-13}$

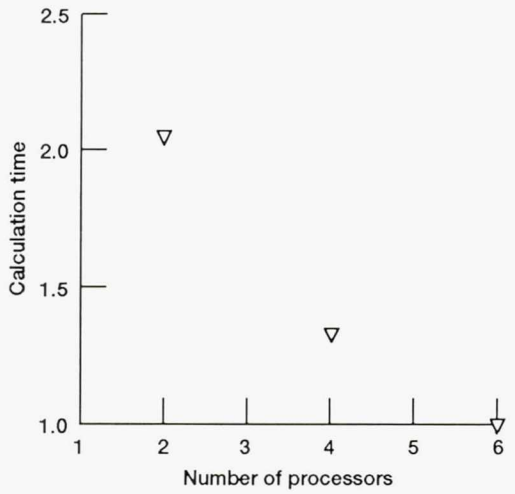

Figure 10.—Calculation time versus number of processors for disk rotating in fluid at rest.

u x-component of velocity
v y-component of velocity

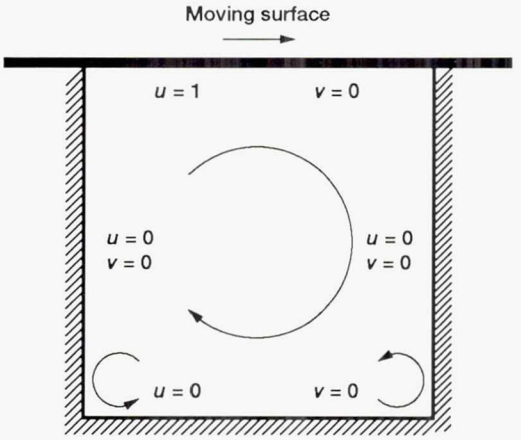

Figure 11.—Driven square cavity showing boundary conditions along with primary and secondary vortices.

The driven square cavity problem was solved over a 50-by-50 finite-difference grid by using four processors in parallel. Compared with using only a single processor to solve the problem serially, a speedup of 2.96 was attained. This speedup corresponds to an efficiency of 74 percent for the four-processor simulation. Graphical outputs for the stream function, the horizontal component of velocity, the vertical component of velocity, and the pressure function were obtained. Typical graphical outputs are displayed in figure 12. The Reynolds number was 100 for the results shown in the figure.

Looking to improve the efficiency being obtained, we determined that an increase in speedup would occur if the inverses of the B diagonal coefficients in the forward and backward reductions were performed in parallel to the rest of the scheme rather than being done serially. The elements of the B coefficient are available substantially before the inverse is required.

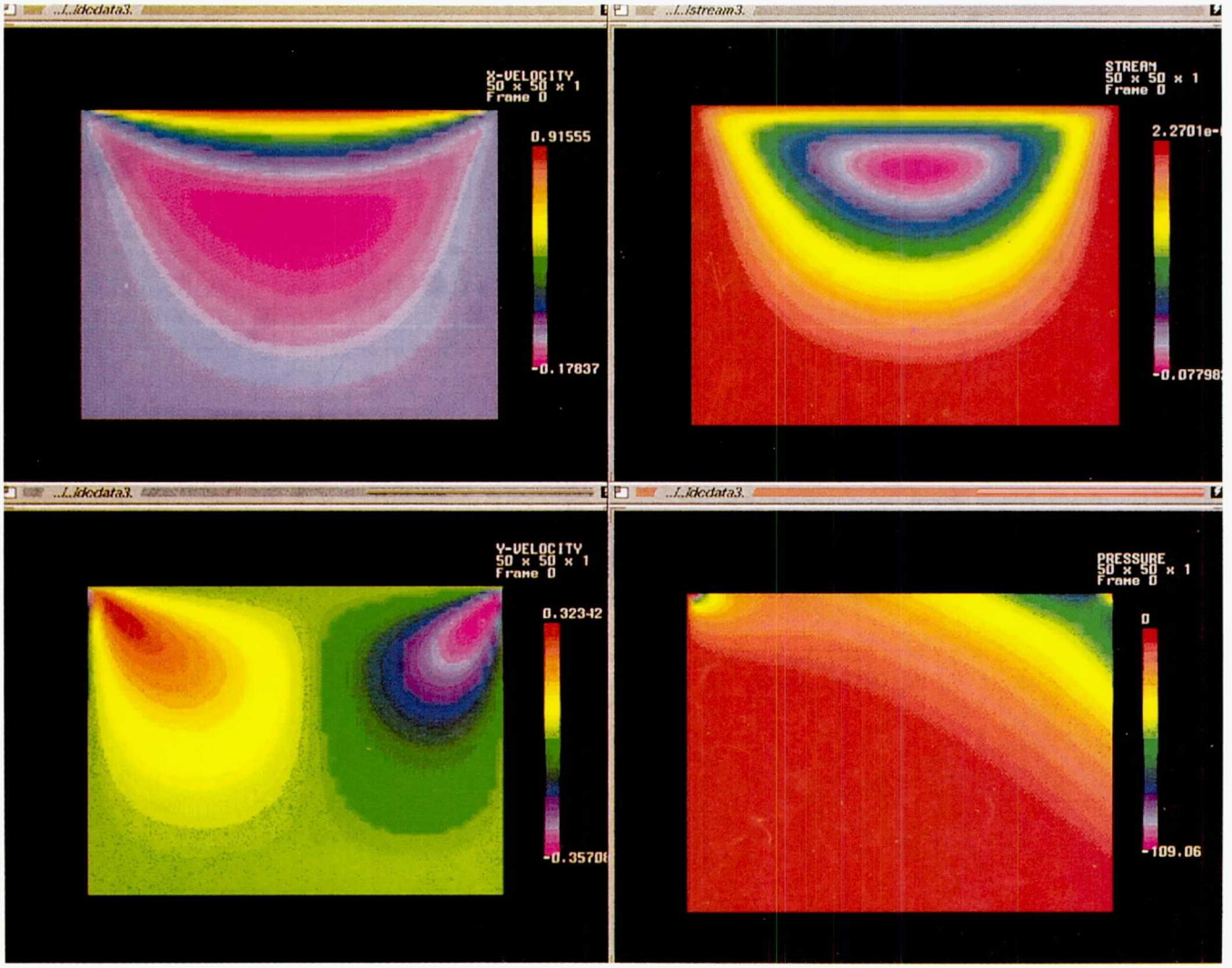

Figure 12.—Driven square cavity. Reynolds number, 100.

Hence, if the inverse were performed in parallel to the main computations, it could be calculated while other portions of the main calculation path were taking place. The inverse would then be ready by the time it was needed in the main calculation path, provided that it could be completed fast enough. And the calculation would be essentally "free" (no overhead), except for the data transfer time. The price for this increased performance is having processors dedicated to the matrix inversion.

Concluding Remarks

A new parallel numerical scheme for solving incompressible steady-state flow has been developed. The algorithm uses a finite-difference approach to solving the Navier-Stokes equations. The code is general and portable. Care was taken to minimize the effects of machine-specific items in order to ease porting of the code between MIMD distributed-memory computers. Machine-specific items, such as COMMON and vector processing, are not used in the code.

The algorithms are scalable and expandable. They may be used with only two processors or with as many processors as are available. The more processors that are available, the more key points may be chosen in the block-tridiagonal system being solved, reducing the work per processor. Work can be balanced among the processors by the choice of key points. Key points need not be equally spaced, although for a homogeneous system equal spacing is a desirable choice for balancing the work load.

Speedup results obtained from applying the parallel algorithm to several different fluid dynamics problems were presented herein. Examining the boundary layer along a flat plate at zero pressure gradient demonstrated the fast convergence rate of the parallel algorithm. In only six iterations the solution converged

to a residual of the order 10^{-12}. Examining the flow in the neighborhood of a disk rotating in a fluid at rest demonstrated that increased speedup is obtained as relative communication time is decreased among the processors used. Following these demonstrations, the parallel algorithm was used to solve for steady-state flow in a driven square cavity. This well-known problem displays the intricate, coupled-flow relationships that are found in complicated fluid dynamics problems. By using a 50-by-50 finite-difference solution grid and four processors of the NASA Lewis Research Center Hypercluster, an efficiency of 74 percent (a speedup of 2.96) was obtained.

References

1. Blech, R.A.; and Arpasi, D.J.: An Approach to Real-Time Simulation Using Parallel Processing. NASA TM–81731, 1981.
2. Milner, E.J.: A Generalized Memory Test Algorithm. NASA TM–82874, 1982.
3. Blech, R.A.; and Arpasi, D.J.: Hardware for a Real-Time Multiprocessor Simulator. NASA TM–83805, 1984.
4. Arpasi, D.J.: RTMPL—A Structured Programming and Documentation Utility for Real-Time Multiprocessor Simulations. NASA TM–83606, 1984.
5. Cole, G.L.: Operating System for a Real-Time Multiprocessor Propulsion System Simulator. NASA TM–83605, 1984.
6. Milner, E.J.; and Arpasi, D.J.: Simulating a Small Turboshaft Engine in a Real-Time Multiprocessor Simulator (RTMPS) Environment. NASA TM–87216, 1986.
7. Lin, A.: Towards Generalization and Optimization of Implicit Methods. Int. J. Numer. Methods Fluids, vol. 5, no. 4, 1985, pp. 357–380.
8. Lin, A.: High Order Three Points Schemes for Boundary Value Problems, Part 1. Linear Problems. SIAM J. Sci. Stat. Commut., vol. 7, no. 3, 1986, pp. 940–958.
9. Lin, A.: Stable Second Order Accurate Iterative Solutions for Second Order Elliptic Problems. Int. J. Numer. Methods Fluids, vol. 9, no. 5, 1989, pp. 583–598.
10. Lin, A.: Parallel Numerical Algorithms for Fluid Dynamics Simulation. AIAA Paper 90–0333, Jan. 1990.
11. Blech, R.A.: The Hypercluster: A Parallel Processing Test-Bed Architecture for Computational Mechanics Applications. NASA TM–89823, 1987.
12. Cole, G.L.; Blech, R.A.; and Quealy, A.: Initial Operating Capability for Hypercluster Parallel Processing Test-Bed. NASA TM–101953, 1989.
13. Lin, A.: Parallel Algorithms for Three Dimensional Flows. Numerical Methods in Laminar and Turbulent Flows, Vol. 5, Part 1, C. Taylor, et al., eds., Pineridge Press, UK, 1987, pp. 48–56.
14. Lin, A.: Fast Parallel Computations of Incompressible Fields. Numerical Methods in Laminar and Turbulent Flow: Proceedings of the Sixth International Conference, C. Taylor ed., Pineridge Press, UK, 1989, pp. 2000–2018.
15. Lin, A.: Parallel Algorithms for Boundary Value Problems, II. Int. J. Parallel Distrib. Comput., vol. 15, 1990, pp. 21–43.
16. Schlichting, H. (J. Kestin, trans.): Boundary Layer Theory. Sixth ed., McGraw-Hill Book Co., Inc., 1966.
17. Bruneau, C.; and Jouron, C.: An Efficient Scheme for Solving Steady State Incompressible Navier-Stokes Equations. J. Comput. Phys., vol. 89, 1990, pp. 389–413.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1993	Technical Memorandum

4. TITLE AND SUBTITLE

Parallel Solution of High-Order Numerical Schemes for Solving Incompressible Flows

5. FUNDING NUMBERS

WU 505–62–52

6. AUTHOR(S)

Edward J. Milner, Avi Lin, May-Fun Liou, and Richard A. Blech

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135–3191

8. PERFORMING ORGANIZATION REPORT NUMBER

E–7345

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, D.C. 20546–0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA TM–4451

11. SUPPLEMENTARY NOTES

Edward J. Milner, NASA Lewis Research Center, Cleveland, Ohio; Avi Lin, University of Pennsylvania, Philadelphia, Pennsylvania 19104, work supported in part by NASA under Space Act Agreement C99066G; May-Fun Liou and Richard A. Blech, NASA Lewis Research Center, Cleveland, Ohio. Responsible person, Edward J. Milner, (216) 433–3656.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 62

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

A new parallel numerical scheme for solving incompressible steady-state flows is presented. The algorithm uses a finite-difference approach to solving the Navier-Stokes equations. The algorithms are scalable and expandable. They may be used with only two processors or with as many processors as are available. The code is general and expandable. Any size grid may be used. Four processors of the NASA Lewis Research Center Hypercluster were used to solve for steady-state flow in a driven square cavity. The Hypercluster was configured in a distributed-memory, hypercube-like architecture. By using a 50-by-50 finite-difference solution grid, an efficiency of 74 percent (a speedup of 2.96) was obtained.

14. SUBJECT TERMS

Parallel processing; Computational fluid dynamics; Computational fluid mechanics; Algorithms

15. NUMBER OF PAGES

16

16. PRICE CODE

A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

National Aeronautics and
Space Administration
Code JTT
Washington, D.C.
20546-0001

Official Business

Penalty for Private Use, $300

BULK RATE
POSTAGE & FEES PAID
NASA
Permit No. G-27

POSTMASTER: If Undeliverable (Section 158 Postal Manual) Do Not Return